Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 11 (2017) 896 – 904

Procedia MANUFACTURING

27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy

Integration and Deployment of a Distributed and Pluggable Industrial Architecture for the PERFoRM Project

Giacomo Angione[a], José Barbosa*[b,c], Frederik Gosewehr[d], Paulo Leitão[b,e], Daniele Massa[a], João Matos[f], Ricardo Silva Peres[f], André Dionisio Rocha[f], Jeffrey Wermann[d]

[a]AEA s.r.l - Loccioni Group, Angeli di Rosora AN, Italy, email: {g.angione, d.massa}@loccioni.com
[b]Polytechnic Institute of Bragança, Campus Sta Apolónia, 5300-253 Bragança, Portugal, email: {jbabrosa, pleitao}@ipb.pt
[c] INESC-TEC, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[d] University of Applied Sciences Emden/Leer, German, email: {frederik.gosewehr, jeffrey.wermann}@hs-emden-leer.de
[e] LIACC – Artificial Intelligence and Computer Science Laboratory, University of Porto, Portugal
[f]CTS UNINOVA, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, email: jp.matos@campus.fct.unl.pt, {ricardo.peres, andre.rocha}@uninova.pt

Abstract

To meet flexibility and reconfigurability requirements, modern production systems need hardware and software solutions which ease the connection and mediation of different and heterogonous industrial cyber-physical components. Following the vision of Industry 4.0, the H2020 PERFoRM project targets, particularly, the seamless reconfiguration of robots and machinery. This paper describes the implementation of a highly flexible, pluggable and distributed architecture solution, focusing on several building blocks, particularly a distributed middleware, a common data model and standard interfaces and technological adapters, which can be used for connecting legacy systems (such as databases) with simulation, visualization and reconfiguration tools.

Keywords: indsutry 4.0, distributed middleware, standard interfaces, technological adapter

* Corresponding author.
E-mail address: jbarbosa@ipb.pt

1. Introduction

A revolution is being currently undertaken in the manufacturing domain. The so called "Industry 4.0" is changing the way how researchers, integrators and companies address the production systems [1]. In fact, several Industry 4.0 related programs have emerged throughout the world, making governments to fund research programs and companies to realise the need to meet these topics in order not to lose competitiveness in a growing market. The main focus of Industry 4.0 is related to the digitization of the whole production ecosystem, covering the levels from the sensors and actuators to the management and strategic ones. To achieve this, several key innovative steps must be undertaken, particularly on the digitization itself, where the Internet of Things technologies are currently playing a major role, interoperability, intelligence and self-* properties, to name a few.

The H2020 PERFoRM (Production harmonizEd Reconfiguration of Flexible Robots and Machinery) project follows this inspiration and aims particularly the seamless reconfiguration of robots and machinery, as well as the plugability and information exchange of existing legacy systems and advanced computational tools exhibiting scheduling, simulation and planning capabilities. For this purpose, a highly flexible, pluggable and distributed system architecture solution was developed, based on a distributed service–oriented industrial middleware, a common data model and standard interfaces and technological adapters, which can be used for connecting legacy systems with simulation, visualization and reconfiguration tools.

This paper describes a practical deployment of the PERFoRM system architecture, considering a Fischertechnik punching cell. The main goal of the usage of such case study cell is to depict how a generic client can access data generated from the cell, as well as the integration of low level legacy components in the PERFoRM system architecture by using technological adapters and the industrial middleware. Therefore, the punching cell is modelled, as the data concerns, a data adapter is created and a middleware integration is performed. Finally, a generic client is created for accessing the data collected from the punching cell.

The rest of the paper is organized as follows: section 2 makes a general overview of the PERFoRM architecture. Section 3 describes the punching cell case study used as the practical exercise for this work and presents its data model instantiation using the PERFoRM specification. Based on this, Section 4 describes the development of an adapter to access data from the punching cell in a PERFoRM-compliant way. Section 5 describes the role of the middleware while Section 6 gives the overall integration details. Section 7 rounds-up the paper with conclusions.

2. Architecture description

The PERFoRM system architecture for the seamless production system reconfiguration [2,3], depicted in Figure 1, is based on a network of distributed hardware devices and software applications and addresses different ISA-95 levels.

These heterogeneous hardware devices and software applications expose their functionalities as services using a distributed service-based integration layer (known as industrial middleware) that ensures a transparent, secure and reliable interconnection. An important innovation of this integration layer is its distributed approach, instead of the traditional centralized ones that can act as a single point of failure as well as a limitation for the system scalability.

Aligned with the general vision for the Industrie 4.0 platform, one of the key challenges that the PERFoRM architecture aims to tackle is the interoperability in real industrial environments, coping with the representation and seamless exchange of data originating from a wide array of entities, often from different, albeit related, actions levels [4]. In this way, the PERFoRM architecture exploits the usage of standard interfaces throughout the whole system, supporting devices, tools and applications, as the main drivers for pluggability and interoperability. This fully exposes and describes services in a unique, standardized and transparent way to enhance the seamless interoperability and pluggability. Therefore, a full specification of the semantics and data flow involved in terms of inputs and outputs required to interact with these elements is necessary. Additionally, and as a support to the standard interfaces, a common data model is adopted, serving as the data exchange format shared between the PERFoRM-compliant architectural elements, covering the semantic needs associated to each entity.

Any innovative architecture, as the one in the PERFoRM project, is only industrially accepted and really adopted if it supports the integration of legacy systems. For this reason, technology adapters are key elements to connect legacy systems to the PERFoRM middleware and to transform the legacy data model into the standard interface, converting legacy components (e.g., an existing database or a robot) into a PERFoRM-compliant component.

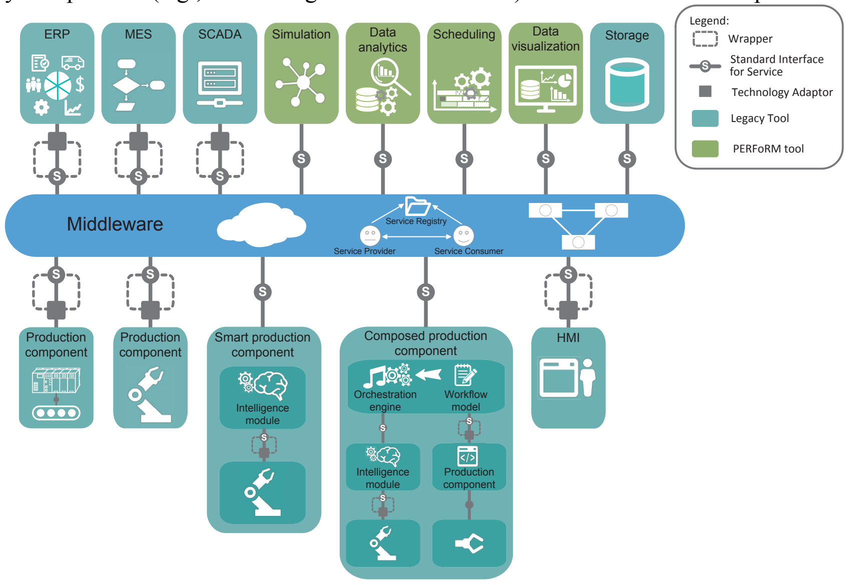

Figure 1 - Overall PERFoRM system architecture.

At a lower level, robots and machinery, to become smarter, need to be empowered with intelligence and higher processing capabilities to run more complex algorithms allowing them to process a high amount of data, producing a valuable analysis to be used when needed (i.e. in a real-time basis) and also supporting the seamless reconfiguration of the system and the achievement of self-* properties, e.g., self-adaptation, self-diagnosis. In fact, the quantity of data being generated in shop floor plants is increasing at a very high rate. The proper analysis, locally (at the edge) and globally (at the cloud), of the collected data assumes a crucial aspect, generating new knowledge that can be used to detect trends, deviations and possible problematic situations beforehand and in a timely manner. At a higher level, the need for tools particularly designed with advanced algorithms and technologies to support the production planning, scheduling and simulation may improve the system performance and reconfigurability. These tools are natively PERFoRM compliant, following the service orientation and using the PERFoRM native interfaces, being inter-connected to the system by means of the industrial middleware.

The aforementioned system architecture features wouldn't be fully exploited if the architecture is not enriched with appropriate mechanisms for the seamless system reconfiguration as well as the introduction of plug-and-produce concepts for a proper "modularity" approach. In PERFoRM, the seamless system reconfiguration is achieved by using the features commonly used in the development of distributed systems, namely those under the technological umbrella of Multi-Agent Systems (MAS) and Service-Oriented Architecture (SOA), particularly service- registry, discovering and composition, which also enhances the plugability, and the proper design, development and deployment of self-* mechanisms. Several self-* properties can be identified in the system architecture, namely self-adaptation, self-diagnosis, self-optimization and self-organization, provided by individual smart production components and advanced tools for planning, scheduling and simulation. The self-organization, leading to the system reconfiguration, emerges from the interaction of the smart production components that have embedded local and global driving self-organization forces implemented using MAS technology. Self-* properties allow the seamless adaptation and reconfiguration of the system; however, in some circumstances the human role is impacted. In these cases, the operators have to be involved in the re-organization process. In fact, the integration of the human in the loop is seen as a key factor to improve flexibility [5], and impose some recommendations for the design of the human-machine and human-human interfaces addressing two different levels: at strategic level, e.g., supporting decision-makers to take strategic decisions, as also at operational level, e.g., supporting operators or maintenance engineers to perform their tasks.

3. Practical application of the PERFoRM architecture

Aiming to exemplify, in a simple manner, the usage of the PERFoRM architecture, a Fischertechnik punching cell is used. This section describes this cell and derives the modelling of its data model.

3.1. Use case description

The Fischertechnik punching cell mimics a simple punching machine and is composed by a set of motors and sensors, as depicted in Figure 2.

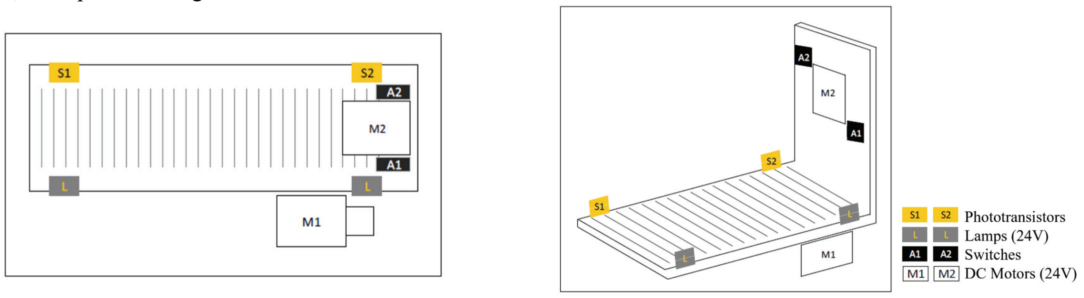

Figure 2 – Overview of the Punching Cell

Two DC motors allow the transportation of the part into the punching tool (motor M1) and the punching down and up movement (motor M2). Two photo-electric switches enable the detection of the part at the beginning and end positions of the conveyor belt. Similarly, two end-course switches allow the control of the up-down movement of the punch tool avoiding the movement outside its limits.

The punching process is relatively simple and consist essentially in a part entering the cell at S1, being then moved to S2, where the punching is performed (controlled by the motor M2). Afterwards, the part is simple moved back along the conveyor (controlled by M1) to S1. The process flow works as follows:

$$S1\ (1) \rightarrow M1F\ (1) \rightarrow S1\ (0) \rightarrow S2\ (1) \rightarrow M1F\ (0) \rightarrow M2D\ (1) \rightarrow A2\ (0) \rightarrow A1\ (1) \rightarrow M2D\ (0) \rightarrow M2U\ (1) \rightarrow A1\ (0) \rightarrow A2\ (1) \rightarrow M2U\ (0) \rightarrow M1T\ (1) \rightarrow S2\ (0) \rightarrow S1\ (1) \rightarrow M1T\ (0)$$

Note that for the use case’s purpose, the granularity target was defined at the motor level, being clear that these two entities, M1 and M2, provide the four main driving skills for the cell, namely the conveyor control, M1F (M1 forward) and M1T (M1 backward), and the punching control, M2U (M2 up) and M2D (M2 down), respectively. In order to enrich the cell's data model, sensorial measurements are also emulated, namely, temperature, heat index, battery voltage and humidity related to motor M1, and temperature and pressure related to the motor M2.

The punching cell is controlled by means of a Schneider Electric M340 PLC (Programmable Logic Controller), where all the I/O signals are available, while the sensorial data, e.g., *Temperature* and *HeatIndex*, is stored directly into a MySQL data base.

3.2. Data model definition

A common data representation format is required in order to integrate and harmonize the different system actors, enabling the seamless exchange of data between them [6]. In this case, these actors are the machinery devices, namely the punching cell itself, the technology adapters and the middleware. As such, this section entails the description of the modelling process for the use case detailed in Section 3.1.

At the low-level, the sensor and I/O data is provided by the PLC, some additional information for both motors is also available in a database, mainly concerning readings such as temperature, humidity, voltages and hydraulic pressure. As such, the overall topological organization of the cell, the model illustrated in Figure 3 was obtained, using the PERFoRM's common data model [6].

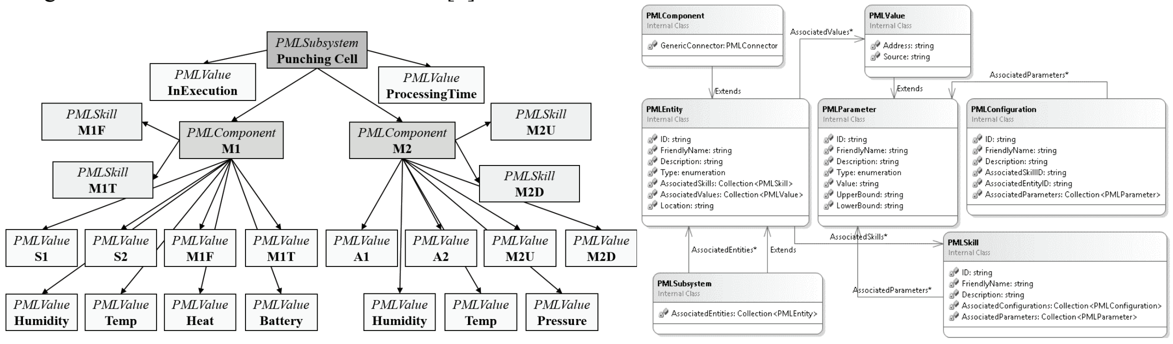

Figure 3 – Use Case Modelling and PERFoRM Data Model Excerpt

The classes applied in the use case are detailed in the data model excerpt represented on the right-hand side of Figure 3. Each one of these classes serves a specific purpose, namely:

- ***PMLComponent*** – Generically abstracts the finest level of granularity, therefore representing all the information required to fully specify a single component in the shop-floor, which may offer a given number of skills (e.g. pick, place, move, weld) and may possess certain values that are relevant to be extracted.
- ***PMLSubsystem*** – Provides a similar functionality, albeit regarding subsystems instead, thus abstracting a group of components and possibly other subsystems.
- ***PMLSkill*** – Exposes the abilities or tasks that a given entity can perform.
- ***PMLValue*** – Enable the basic representation of information pertaining to data at the machinery level, namely in terms of shop-floor data to be extracted from various sources.

Taking these aspects into account, to fully enable the seamless integration of both intelligent and legacy systems, there is a need to couple this common representation with technology adapters. These adapters will be explained in further detail in Section 4.

4. Adapting legacy system by means of technological adapters

At its essence, the adapter's function is to convert legacy data into the PERFoRM's data model and to implement the PERFoRM's standard interfaces. Middleware solutions and technological adapters, like the ones described in this paper, permit the interconnection of heterogeneous legacy hardware devices, e.g. robots and the respective controllers, and software applications, e.g. databases, SCADA applications and other management, analytics and logistics tools.

In the use case, described in Section 3.1, the database containing readings about temperature, humidity, voltage and hydraulic pressure of the punching cell, constitutes a legacy system for which a technological adapter is needed.

The role of the adapter is to retrieve the values from the database and translate them according to the PERFoRM's data model, specfified in the previous section. In particular, the adapter performs the following activities:

1. Implement the PERFoRM's standard interface and methods.
2. Connect to the database.
3. Send queries and update statements to the database.
4. Retrieve the results of the queries from the database.
5. Transform the results according to the PERFoRM's data model.

The connection to the database is facilitated by the use of the JDBC (Java Database Connectivity) interface that provides a transparent connectivity to heterogeneous databases. Since the use case database is MySQL, the corresponding driver must be used to access to the data source. The simple code fragment, depicted below, gives an example of the database adapter implementation:

```
public class DB_Adaptor implements PMLInterface {
        public DB_Adaptor() {}
        public boolean init() {
                try {
                        Class<?> driver_class = Class.forName("DriverName");
                        Driver driver = (Driver) driver_class.newInstance();
                        DriverManager.registerDriver(driver);
                        con = DriverManager.getConnection(url, user, password);
                        return true;
                } catch (Exception exc) {
                        return false;
                }
        }
        public boolean close() {
                try {
                        con.close();
                        return true;
                } catch (Exception exc) {
                        return false;
                }
        }
        @Override
        public PMLValue getValue(String valueID) {
                try {
                        Statement stmt = con.createStatement();
                        ResultSet rs = stmt.executeQuery(query);
                        rs.next();
                        float float_val = rs.getFloat(valueID);
                        PMLValue val = new PMLValue();
                        val.setID(valueID);
                        val.setAddress(connectionURL);
                        val.setSource(Source.DB);
                        val.setValueType(Type.FLOAT);
                        val.setValue(""+float_val);
                        return val;
                } catch (Exception exc) {
                        return null;
                }
        }
}
```

Figure 4 – Database Adapter Implementation Code

The *init* method of the database adapter instantiates a *DriverManager* object to connect to the database driver and logs into the database. The *getValue* method instantiates a *Statement* object that carries an SQL language query to the database, instantiates a *ResultSet* object that retrieves the result of the query, and transforms this result in the PERFORM's data model, instantiating and populating a *PMLValue* object. The *close* method closes the connection.

The Java code for the MySQL database adapter, presented in this section, can then be easily transformed into a web service as described in Section 6. The creation of the web service allows a smooth integration of the adapter into the PERFoRM's middleware, which takes care of the conversion of Java objects to and from JSON strings and permits the service registration and discovery.

5. Communicating blocks: a middleware solution

One of the goals of the architecture foreseen within PERFoRM is to find a common integration platform for all the PERFoRM compliant components. Given that they follow the specifications and use the data models described in Section 3.1, any system should be able to communicate and interact with each other by using a so-called Middleware. Instead of having to implement specific interfaces to each other component, it is now only necessary to interface with one component – the Middleware. This Middleware will route the incoming messages to the right recipient(s), also translating between different communication protocols. For the components involved in the communication, the Middleware is basically transparent. They communicate to the Middleware in the same way they would communicate with the communication partner directly, just without having to worry about adapting the communication technology to fit the communication partner. Additionally, the Middleware is providing other services, namely discovery and data processing mechanisms. Each component should be able to register the services they provide within the Middleware, so that other connected systems will be able to dynamically discover it without the need to know about them from the beginning. The data processing mechanism allows the Middleware to store the data coming through and manipulating it, if necessary. This can for example be used to transform data from one data format to another.

Multiple commercial solutions which provide the functionality of a Middleware already exist, often focusing on specific areas of application. For example, the communication protocols and the amount and types of data on a business level are usually very different to the ones used on a production system. Therefore, a lot of existing solutions are specialized to one or the other and sometimes are lacking important features. To fill in these gaps and to make the existing Middleware solutions comply with the PERFoRM architecture, additional development is necessary. From the perspective of the tools and other systems connecting to the Middleware it should also not matter, how it is implemented specifically. To target these issues, Apache ServiceMix is used as a tool to integrate existing Middleware solutions to comply the specifications set within PERFoRM, since it provides a lot of useful tools for integration and is the base for several solutions. Additionally, ServiceMix can also act as a Middleware on its own.

For the practical application in the use case described in Section 3, a full implementation of a commercial Middleware solution is not feasible, due to the costs and size of such systems. To get an easy open source implementation of a PERFoRM-compliant Middleware, which fulfils the purpose of demonstrating how the Middleware works and how other software can interface to it, a stand-alone Apache ServiceMix approach is sufficient.

The Middleware implementation itself is independent of the actual application. Therefore, the only issues necessary to be adjusted to the specific application are: i) integration of the adapters, ii) implementation and integration of a service consumer (client), iii) integration of the PERFoRM data model, and iv) configuration of the routing. [7]

6. System integration

For the integration of the adapters and the implementation of a client, Apache CXF is used to create REST based services. A REST server is set up in Java to implement a service to retrieve the value “pressureBME” from the Punching Cell. This service is linked to the MySQL database adapter described in Section 4. Another service provides the PERFoRMML data model as a JSON object. These services can be consumed using REST, implemented in Java within Apache CXF. For the routing of the messages within the Middleware itself, Apache Camel is used.

Building such service provider, which encapsulates the adapter implementation, follows a few defined steps and is based on Java JAXRS 2.0 (defined in JSR 339) servlet integration, here on the implementation used in Apache CXF 3.1.9. Together with Spring-Boot, which is the self-contained Java runtime container in which every dependency is integrated in, e.g., the webserver the services are running on, the combination of both technologies creates a way of rapid prototyping production ready services which also include their own integration and unit test. This ensures that besides a JRE environment, no other system dependencies are required from the service itself.

The starting point of each project is based on the maven archetype **org.apache.cxf.archetype/cxf-jaxrs-service**, which creates the boilerplate code for the service integration. Besides the default dependencies, the PERFoRM service provider also includes Swagger and WADL (Web Application Description Language) as integrated API description/documentation into each service. In this way, the documentation of a service is auto-generated during runtime and always up to date with the actual implementation. Besides that, the WADL description can directly be used to register a provider with the middleware to expose the available services via the common API. To interface with a service, XML and JSON are used and parsed automatically through JAXB into object files which decreases the amount of integration work necessary. Using common and shared service interface definition files, a Java based client can easily connect via a CXF client proxy without knowing the correct path and additionally use the remote API like a RPC call. In summary, the sequence of steps to create the service is:

1. Create the maven project from archetype.
2. Modify the Pom.xml to integrate the missing dependencies and integrate Spring-boot build config.
3. Define a service interface class with CXF Annotations based on the method names of the adapter to implement.
4. Implement the defined interface into the class which implements the adapter.

5. Define a Spring-Boot JAXRS Service Bean starter (the Spring-Boot main configuration) which wires the needed connections to the container runtime.
6. Define the WADL registration method, which, after bootstrapping Spring-Boot, automatically probes the network for the middleware, and, if available, registers the AML description with both the WADL and PML.
7. Define the integration tests using the CXF client proxy and Junit/Spring-Boot-Test. This is an important step if continuous integration is mandatory. The test simulates a running Spring-Boot instance running the service.
8. Package to automatically test before the deployment.

The actual implementation of a service skeleton has a very low overhead through the use of dependency injection and runtime annotation parsing. To the programmer, the service is only the annotation made to the interface or method, depending how the client should use the service (using an interface is required to use the JAXRSClientProxy) while the heavy lifting is handled by the underlying libraries creating this services during runtime and wiring them to the embedded webserver, here jetty. An example how to transform an arbitrary Java method, here for the code implementing the database adapter described by Figure 5.

Original Code implementing Figure 4

```
@Override
public String retrievePMLValue(String ID) {
  String retval = "null";
  PMLInterfaceImpl parser = new    PMLInterfaceImpl();
  if (parser.init()) {
    PMLValue val = parser.getData(ID);
    if (val != null) {
      retval = val.getValue();
    } else {
      retval = "valueID not valid!";
    }
    parser.close();
  }
  return retval;
}
```

Extended via JAX-RS annotations to create web service

```
@GET
@Consumes("text/plain")
@Produces("text/plain")
@Path("/retrieve_pml_value/{id}")
@Override
public String retrievePMLValue(@PathParam("id") String ID
) {
  String retval = "null";
  PMLInterfaceImpl parser = new    PMLInterfaceImpl();

 …
  return retval;
}
```

Figure 5– CXF JAX-RS web service migration of arbitrary Java code

7. Conclusion

The manufacturing domain is assisting to a technological and paradigm revolution, commonly coined as Industry 4.0 and where digitization, system interoperability and pluggablity are at the cornerstone of the applied concepts. The H2020 PERFoRM project follows these steps, proposing an innovative system architecture to achieve these key features. PERFoRM promotes the usage of a modular concept where modules are communicating by means of standard interfaces and applying a common data model. The connection of legacy components is enabled by the development of adaptors, responsible for the native data format into the PERFoRM data model. Finally, this modular concept is interconnected by means of a industrial middleware that offers the necessary communication infrastructure.

This paper depicted this system architecture and presented a simple, but exemplificative, practical example of the integration between existing low level PLC data and a system consumer, for a Fischertechnik punching cell. For this purpose, the paper details the instantiation of the PERFoRM data model, the creation of the MySQL database adapter and the setup of a client service that was interconnected to the MySQL database by using the industrial middleware.

Future steps are devoted to the full-range deployment and assessment of the system architecture in use cases.

Acknowledgements

This project has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No 680435.

References

[1] H. Kagermann andW.Wahlster and J. Helbig: Securing the future of German manufacturing industry: Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Tech. rep., ACATECH – German National Academy of Science and Engineering (2013)
[2] Deliverable D2.2, "Definition of the System Architecture", PERFoRM project, 28th September 2016.
[3] P. Leitão, J. Barbosa, A. Pereira, J. Barata, A.W. Colombo, "Specification of the PERFoRM architecture for seamless production system reconfiguration", Proceedings of the 42nd Annual Conference of IEEE Industrial Electronics Society (IECON'16), October 24-27, Firenze, Italy, 2016.
[4] H. Bauer, C. Baur, G. Camplone, and et. al., "Industry 4.0: How to Navigate Digitization of the Manufacturing Sector", Technical report, McKinsey Digital, 2015.
[5] Deliverable D2.1, "Guidelines for Seamless Integration of Humans as Flexibility Driver in Flexible Production Systems", PERFoRM project, 11th March 2016.
[6] Deliverable D2.3, "Specification of the Generic Interfaces for Machinery, Control Systems and Data Backbone", PERFoRM project, 23rd of January 2017.
[7] Deliverable D2.4, "Industrial Manufacturing Middleware: Specification, prototype implementation and validation", PERFoRM project, 31st of March 2017.